

Chapel: Productive Parallel Programming at all Scales

Brad Chamberlain, Advanced Programming Team, HPE

SSEC, eScience Institute, University of Washington

August 25, 2026

What is Chapel?

Chapel: A modern parallel programming language

- Portable & scalable
- Open-source & collaborative
 - an HPSF / Linux Foundation project

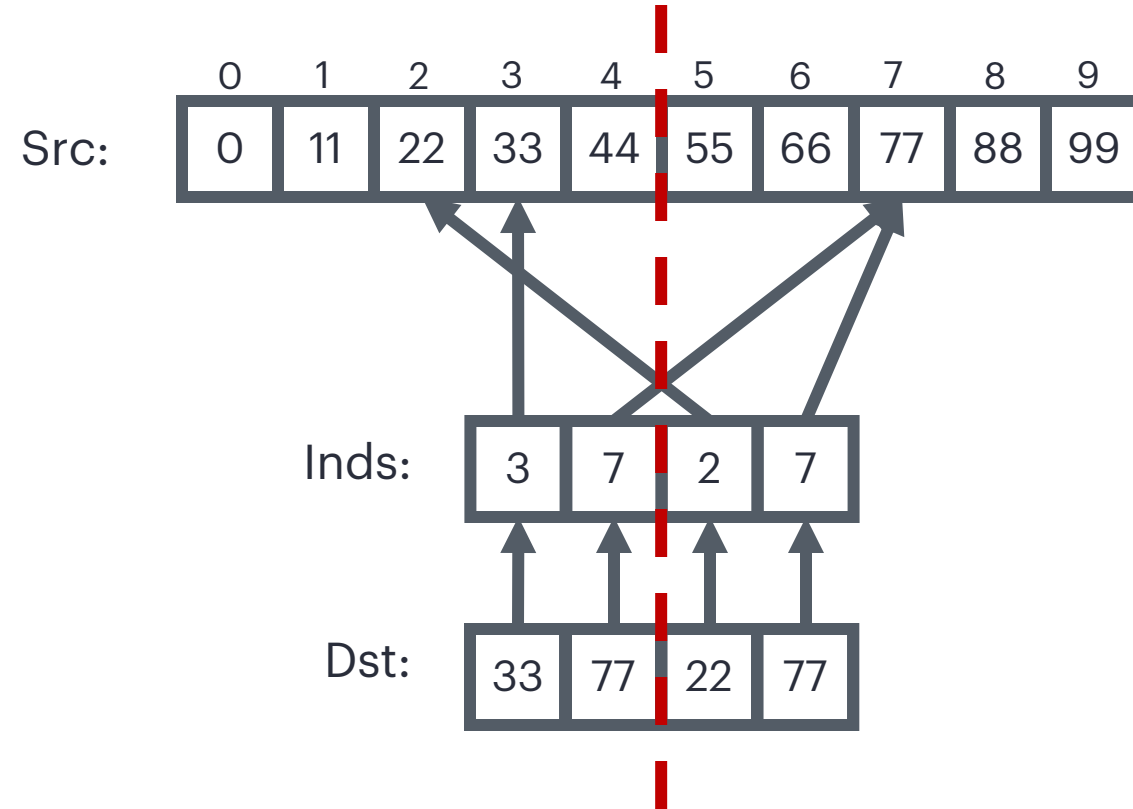
Goals:

- Support general parallel programming
- Make parallel programming at scale far more productive



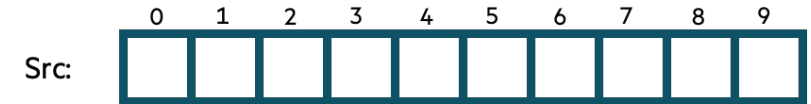
Chapel By Example: Bale Index Gather

Bale Index Gather (IG): In Pictures



Bale IG in Chapel: Scalar and Array Declarations

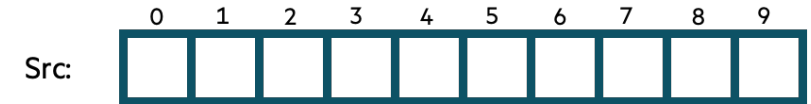
```
config const n = 10,  
            m = 4;  
  
var Src: [0..  
    n] int,  
    Inds, Dst: [0..  
    m] int;
```



\$

Bale IG in Chapel: Compiling

```
config const n = 10,  
            m = 4;  
  
var Src: [0..  
    Inds, Dst: [0..  
    int;
```



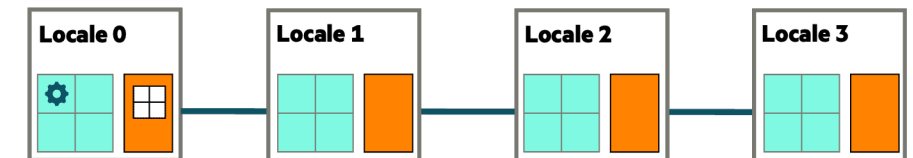
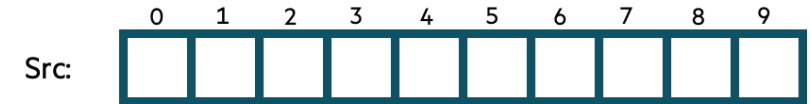
```
$ chpl bale-ig.chpl
```

```
$
```

Bale IG in Chapel: Executing

```
config const n = 10,  
            m = 4;  
  
var Src: [0..  
    Inds, Dst: [0..  
    int;
```

```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



Bale IG in Chapel: Executing, Overriding Configs

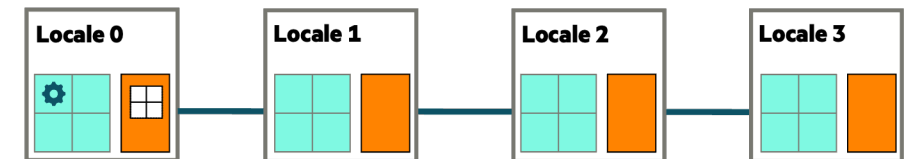
```
config const n = 10,  
            m = 4;  
  
var Src: [0..<n] int,  
    Inds, Dst: [0..<m] int;
```

```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4 --n=1_000_000 --m=1_000_000  
$
```

Src: 

Inds: 

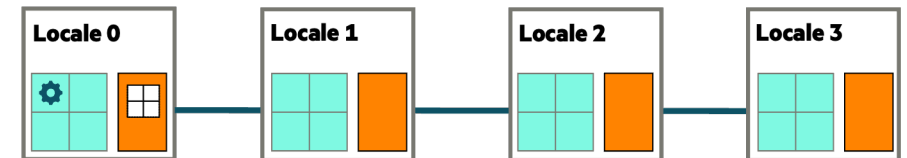
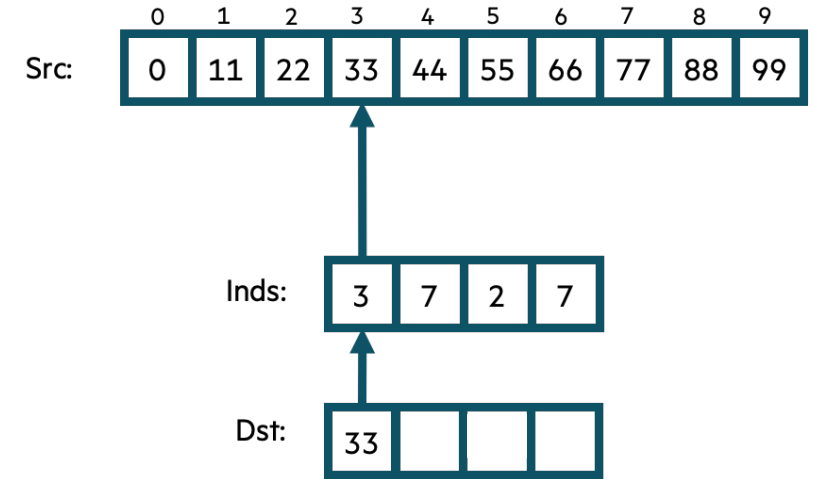
Dst: 



Bale IG in Chapel: Serial, Zippered Version

```
config const n = 10,  
            m = 4;  
  
var Src: [0..  
    Inds, Dst: [0..  
...  
for (d, i) in zip(Dst, Inds) do  
    d = Src[i];
```

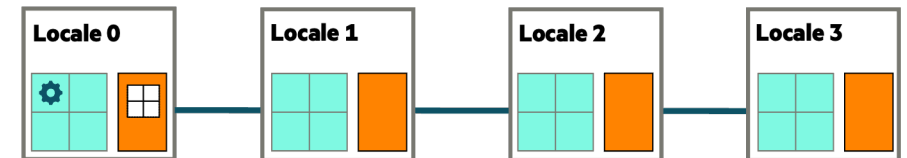
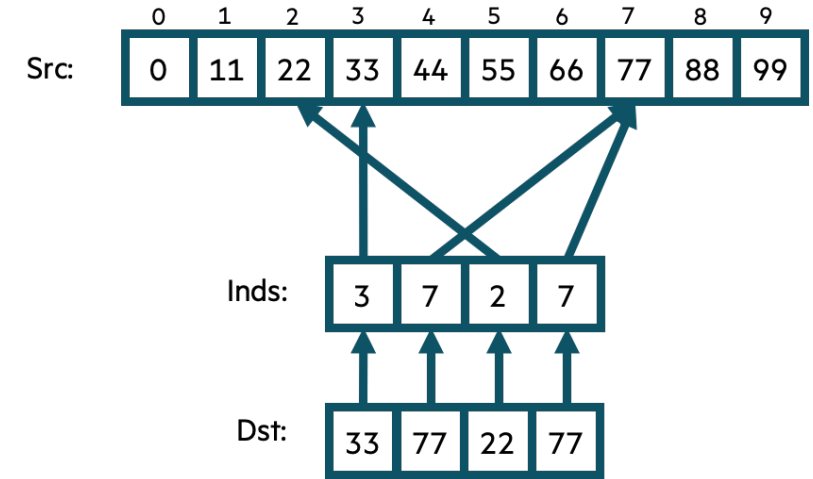
```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



Bale IG in Chapel: Parallel, Zippered Version (vectorized)

```
config const n = 10,  
            m = 4;  
  
var Src: [0..  
    Inds, Dst: [0..  
...  
foreach (d, i) in zip(Dst, Inds) do  
    d = Src[i];
```

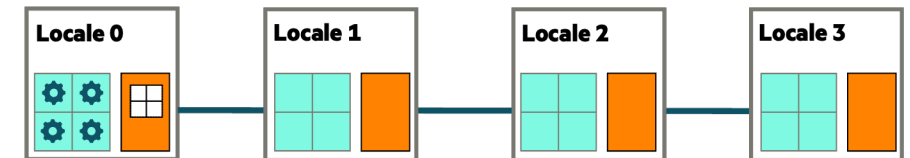
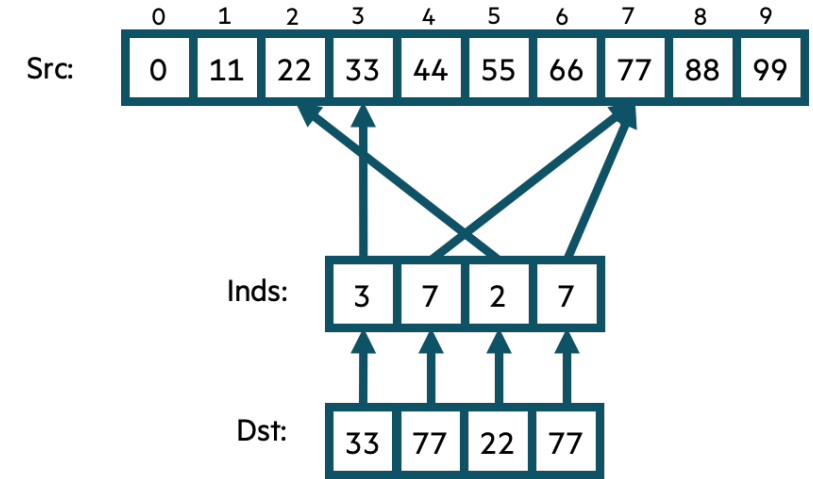
```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



Bale IG in Chapel: Parallel, Zippered Version (multicore)

```
config const n = 10,  
            m = 4;  
  
var Src: [0..<n] int,  
    Inds, Dst: [0..<m] int;  
...  
forall (d, i) in zip(Dst, Inds) do  
    d = Src[i];
```

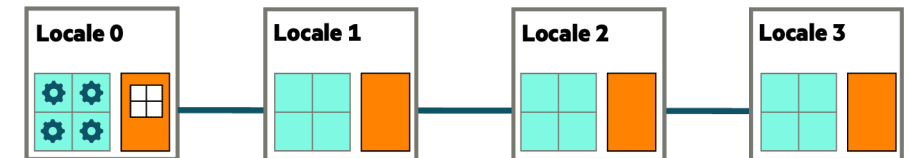
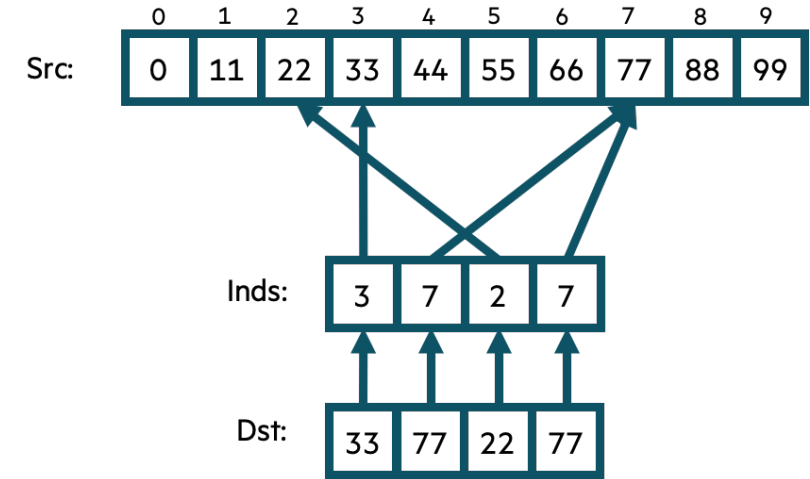
```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



Bale IG in Chapel: Parallel Promoted Version (equivalent to previous version)

```
config const n = 10,  
            m = 4;  
  
var Src: [0..  
    Inds, Dst: [0..  
...  
Dst = Src[Inds];
```

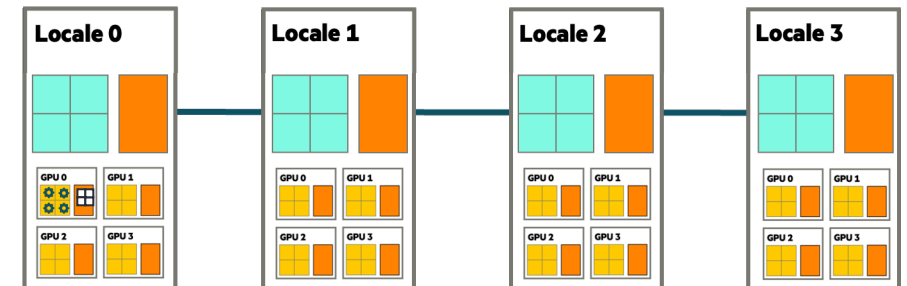
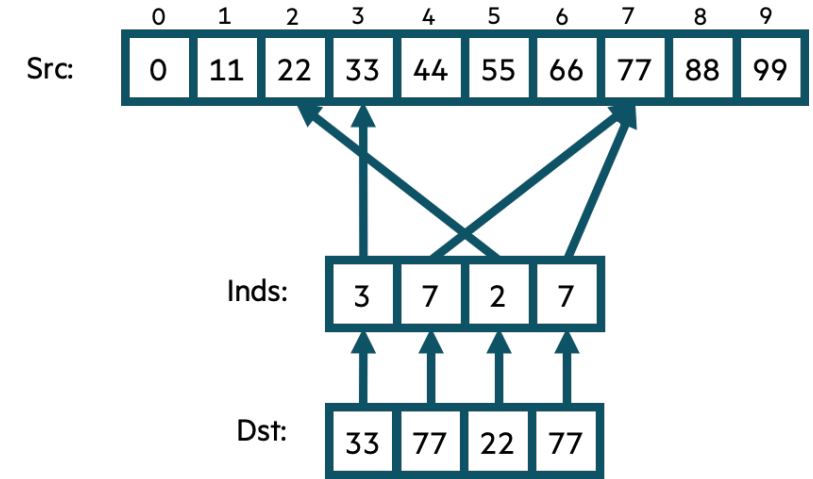
```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



Bale IG in Chapel: Parallel, Zippered Version (single GPU)

```
config const n = 10,  
            m = 4;  
  
on here.gpus[0] {  
  var Src: [0..  
    Inds, Dst: [0..  
  ...  
  Dst = Src[Inds];  
}
```

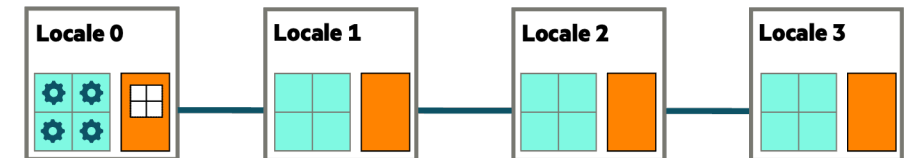
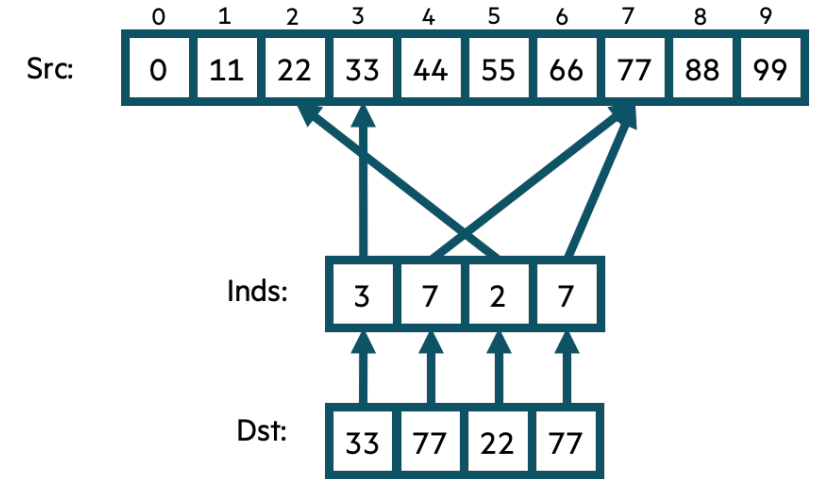
```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



Bale IG in Chapel: Parallel, Zippered Version (Multicore, again)

```
config const n = 10,  
            m = 4;  
  
var Src: [0..<n] int,  
    Inds, Dst: [0..<m] int;  
...  
forall (d, i) in zip(Dst, Inds) do  
    d = Src[i];
```

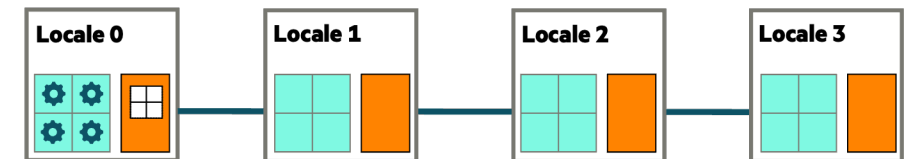
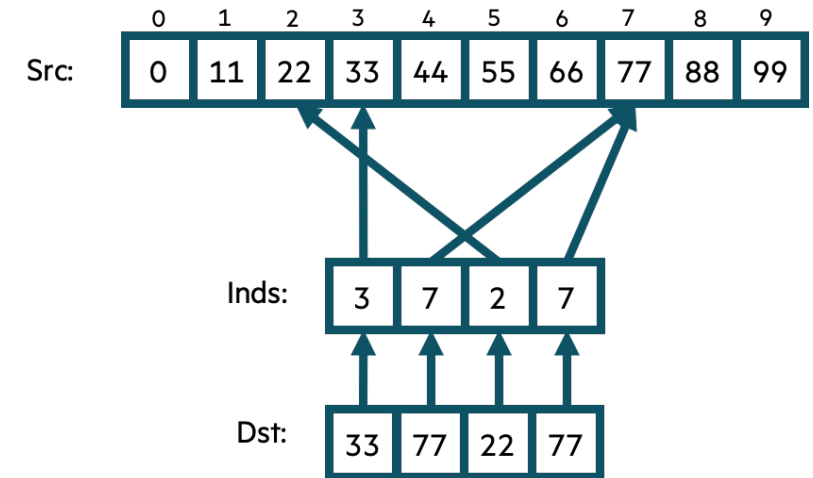
```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



Bale IG in Chapel: Parallel, Zippered Version with Named Domains (Multicore)

```
config const n = 10,  
            m = 4;  
  
const SrcInds = {0..  
  DstInds = {0..  
  
var Src: [SrcInds] int,  
    Inds, Dst: [DstInds] int;  
...  
forall (d, i) in zip(Dst, Inds) do  
  d = Src[i];
```

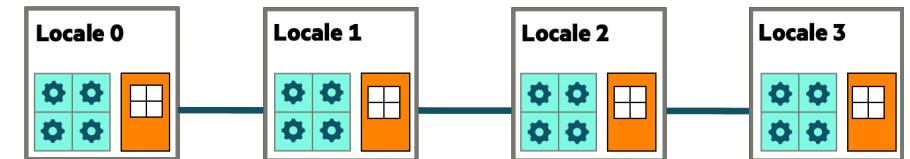
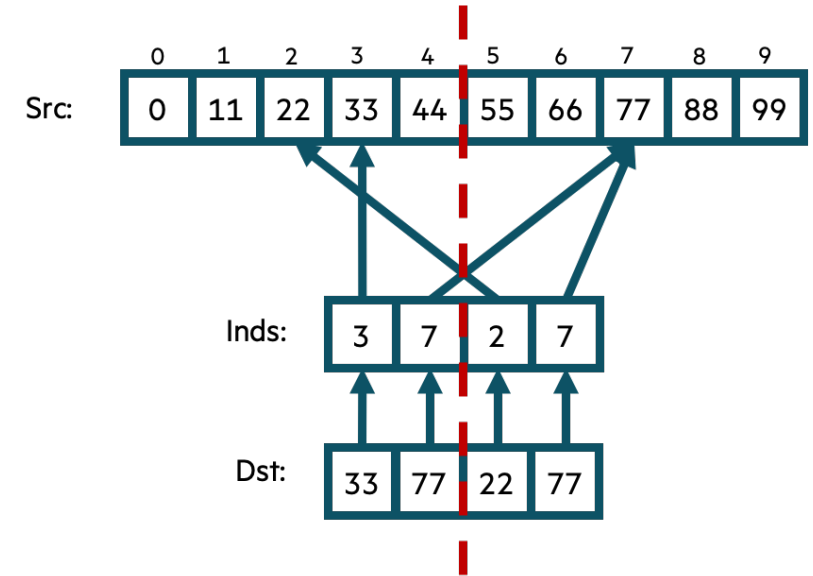
```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



Bale IG in Chapel: Distributed Parallel Version

```
use BlockDist;  
  
config const n = 10,  
            m = 4;  
  
const SrcInds = blockDist.createDomain(0..  
    DstInds = blockDist.createDomain(0..  
  
var Src: [SrcInds] int,  
    Inds, Dst: [DstInds] int;  
...  
forall (d, i) in zip(Dst, Inds) do  
    d = Src[i];
```

```
$ chpl bale-ig.chpl  
$ ./bale-ig -nl 4  
$
```



Bale IG in Chapel: Distributed Parallel Version on HPE Cray EX

```
use BlockDist;

config const n = 10,
            m = 4;

const SrcInds = blockDist.createDomain(0..<n),
       DstInds = blockDist.createDomain(0..<m);

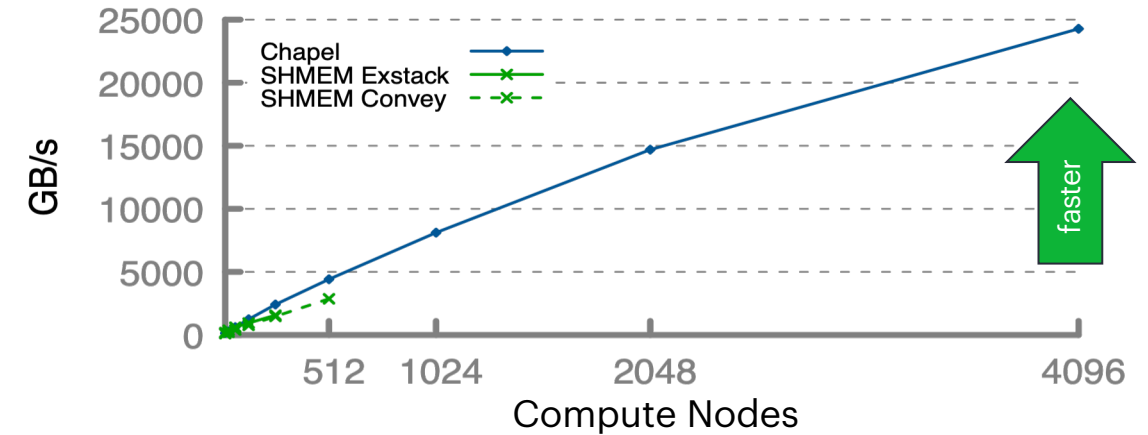
var Src: [SrcInds] int,
     Inds, Dst: [DstInds] int;

...
forall (d, i) in zip(Dst, Inds) do
    d = Src[i];
```

```
$ chpl bale-ig.chpl --fast --auto-aggregation
$ ./bale-ig -nl 4096 --n=... --m=...
$
```

Bale Indexgather Performance

HPE Cray EX (Slingshot-11)



Bale IG in Chapel vs. SHMEM on HPE Cray EX

Chapel

```
forall (d, i) in zip(Dst, Inds) do
    d = Src[i];
```

SHMEM (Exstack version)

```
i=0;
while( exstack_proceed(ex, (i==l_num_req)) ) {
    i0 = i;
    while(i < l_num_req) {
        l_indx = pckindx[i] >> 16;
        pe = pckindx[i] & 0xffff;
        if(!exstack_push(ex, &l_indx, pe))
            break;
        i++;
    }

    exstack_exchange(ex);

    while(exstack_pop(ex, &idx, &fromth)) {
        idx = ltable[idx];
        exstack_push(ex, &idx, fromth);
    }
    lgp_barrier();
    exstack_exchange(ex);

    for(j=i0; j<i; j++) {
        fromth = pckindx[j] & 0xffff;
        exstack_pop_thread(ex, &idx, (uint64_t)fromth);
        tgt[j] = idx;
    }
    lgp_barrier();
}
```

SHMEM (Conveyors version)

```
i = 0;
while (more = convey_advance(requests, (i == l_num_req)),
       more | convey_advance(replies, !more)) {

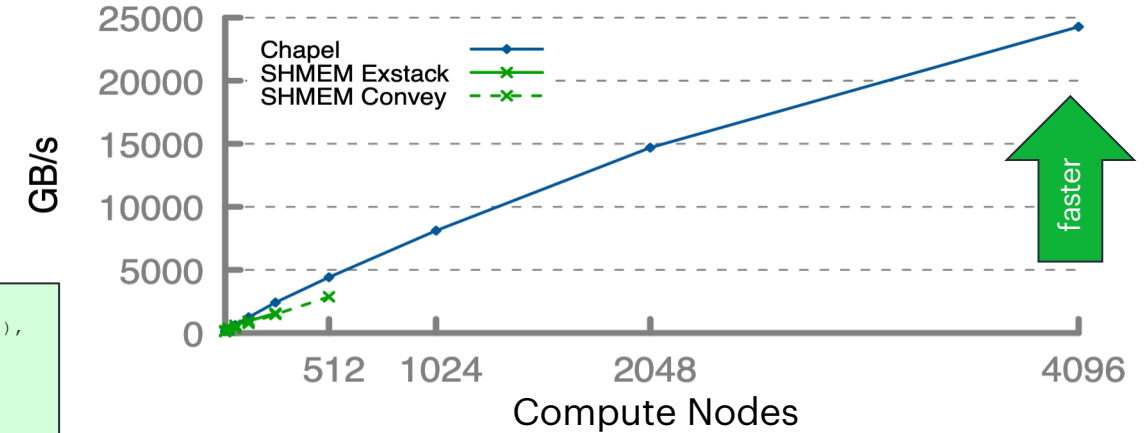
    for (; i < l_num_req; i++) {
        pkg.idx = i;
        pkg.val = pckindx[i] >> 16;
        pe = pckindx[i] & 0xffff;
        if (!convey_push(requests, &pkg, pe))
            break;
    }

    while (convey_pull(requests, ptr, &from) == convey_OK) {
        pkg.idx = ptr->idx;
        pkg.val = ltable[ptr->val];
        if (!convey_push(replies, &pkg, from)) {
            convey_unpull(requests);
            break;
        }
    }

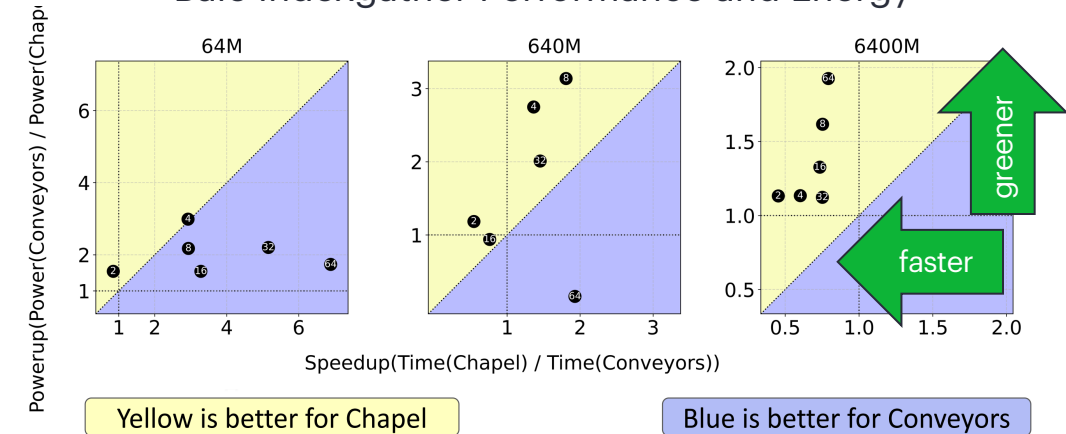
    while (convey_pull(replies, ptr, NULL) == convey_OK)
        tgt[ptr->idx] = ptr->val;
}
```

Bale Indexgather Performance

HPE Cray EX (Slingshot-11)



Bale Indexgather Performance and Energy



There's lots more to Chapel...

Additional parallel features:

- multidimensional, associative, and sparse arrays
- explicit task-based parallelism
- variables for synchronized data sharing
- reduction and scan operators
- ...

Additional scalability features:

- explicit control over placement of data and tasks
- a global namespace
- the ability to reason about location
- ...

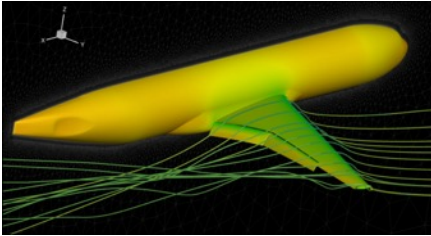
Additional base language features:

- OOP (value- and reference-based)
- modules for namespacing
- overloading, rich arguments, generics
- type inference
- memory safety
- ...



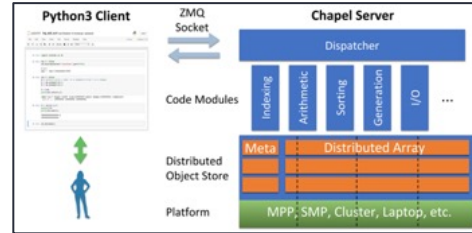
Applications of Chapel / Arkouda

Applications of Chapel



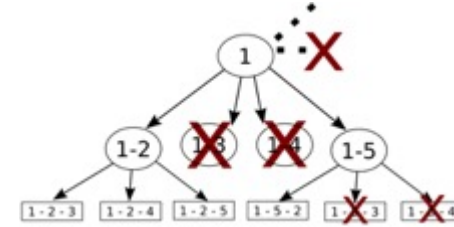
CHAMPS: 3D Unstructured CFD

Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



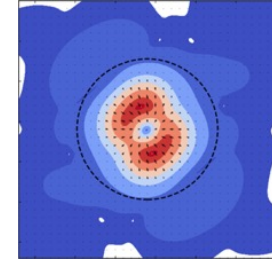
Arkouda: Interactive Data Science at Massive Scale

Mike Merrill, Bill Reus, et al.
U.S. DoD



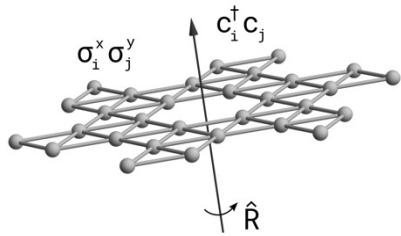
ChOp: Chapel-based Optimization

T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



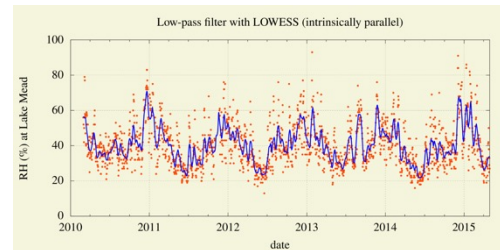
ChplUltra: Simulating Ultralight Dark Matter

Nikhil Padmanabhan, J. Luna Zagorac, et al.
Yale University et al.



Lattice-Symmetries: a Quantum Many-Body Toolbox

Tom Westerhout
Radboud University



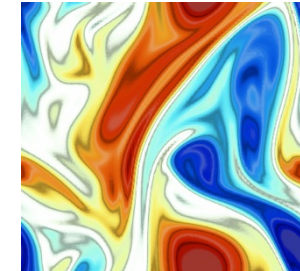
Desk dot chpl: Utilities for Environmental Eng.

Nelson Luis Dias
The Federal University of Paraná, Brazil



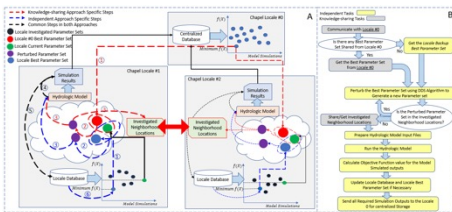
RapidQ: Mapping Coral Biodiversity

Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



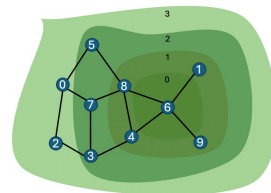
ChapQG: Layered Quasigeostrophic CFD

Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.



Chapel-based Hydrological Model Calibration

Marjan Asgari et al.
University of Guelph



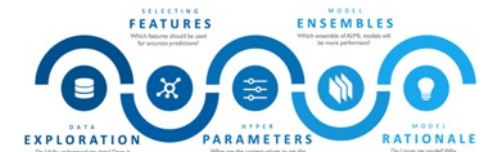
Arachne Graph Analytics

Bader, Du, Rodriguez, et al.
New Jersey Institute of Technology



Modeling Ocean Carbon Dioxide Removal

Scott Bachman Brandon Neth, et al.
[C]Worthy



CrayAI HyperParameter Optimization (HPO)

Ben Albrecht et al.
Cray Inc. / HPE

[images provided by their respective teams and used with permission]

Data Science In Python at scale?

Motivation: Imagine you have...

- ...Python-based data scientists
- ...HPC-scale data science problems to solve
- ...access to HPC systems



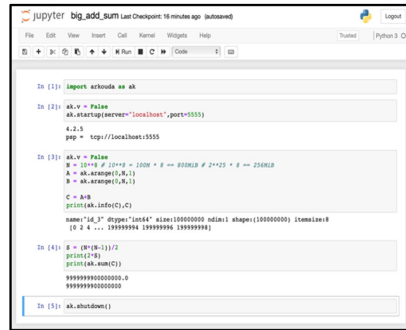
How will you enable your Python programmers to get their work done?

What is Arkouda?

Q: “What is Arkouda?”



Arkouda Client
(written in Python)

A screenshot of a Jupyter Notebook interface. The title bar says 'jupyter big_add_sum Last Checkpoint: 10 minutes ago (auto-saved)'. The code in the notebook includes:

```
In [1]: import arkouda as ak
In [2]: ak.N = False
      ak.startUpServer("localhost", port=5555)
      4.2.5
      prep = http://localhost:5555
In [3]: ak.N = False
      N = 10**8 # 2**16 = 65536 * 2**12 = 8192 * 8192 * 2**12 = 256 * 8192
      A = ak.arange(N, dtype=ak.uint64)
      B = ak.arange(N, dtype=ak.uint64)
      C = A+B
      print(ak.info(C))
      name: 'A_B' dtype: 'uint64' size: 100000000 ndim: 1 shape: (100000000,) itemsize: 8
      [0 2 4 ... 199999998 199999999]
In [4]: S = ak.zeros(N, dtype=ak.float64)
      print(S)
      9999999999999999.0
      9999999999999999.0
In [5]: ak.shutdown()
```



User writes Python code
making familiar NumPy/Pandas calls

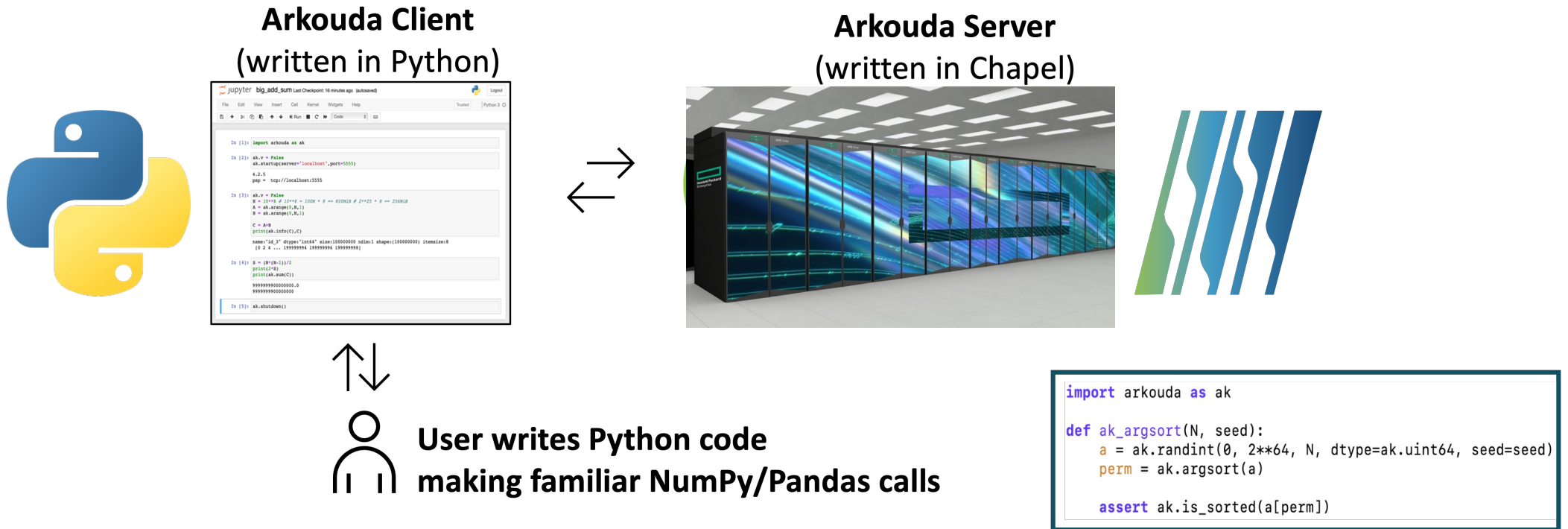
```
import arkouda as ak

def ak_argsort(N, seed):
    a = ak.randint(0, 2**64, N, dtype=ak.uint64, seed=seed)
    perm = ak.argsort(a)

    assert ak.is_sorted(a[perm])
```

What is Arkouda?

Q: “What is Arkouda?”



A1: “A scalable version of NumPy / Pandas routines for data scientists”

A2: “A framework for driving supercomputers interactively from Python”

Performance and Productivity: Arkouda Argsort

HPE Cray EX

- Slingshot-11 network (200 Gb/s)
- 8192 compute nodes
- 256 TiB of 8-byte values
- ~8500 GiB/s (~31 seconds)

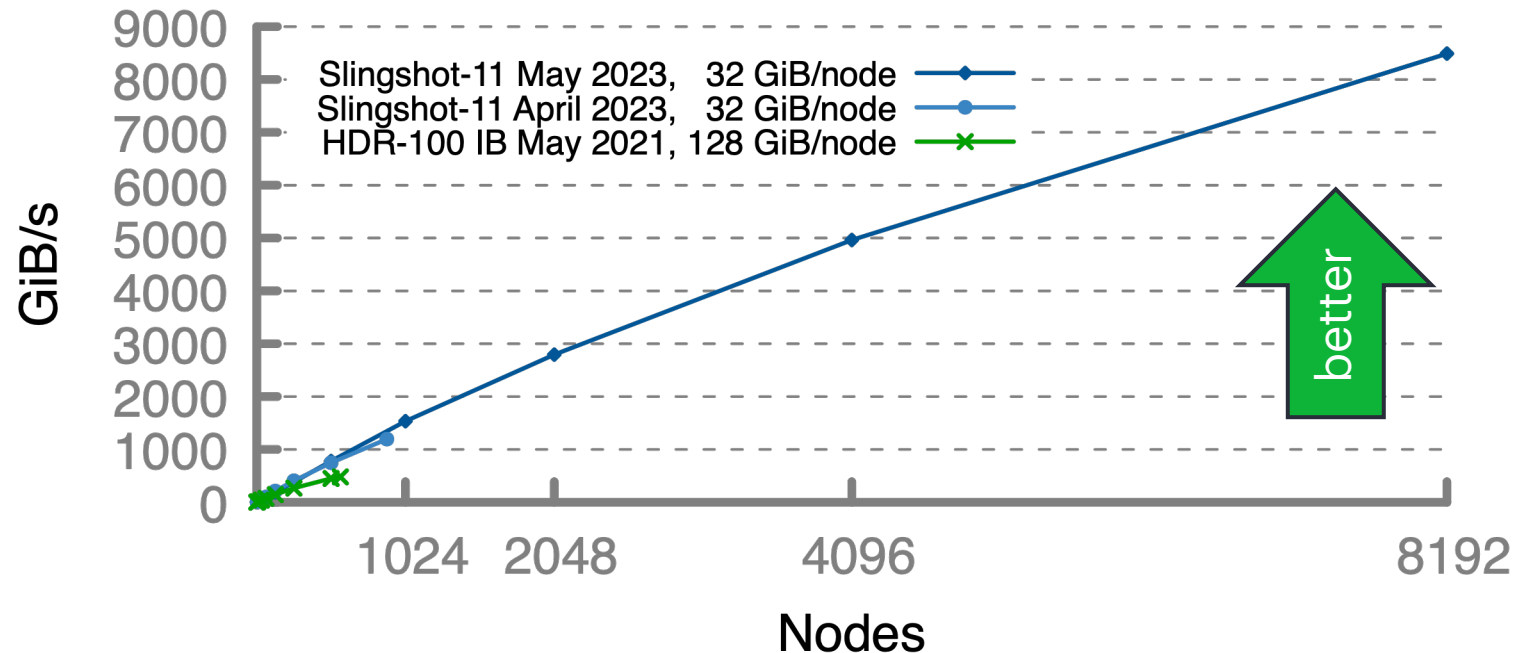
HPE Cray EX

- Slingshot-11 network (200 Gb/s)
- 896 compute nodes
- 28 TiB of 8-byte values
- ~1200 GiB/s (~24 seconds)

HPE Apollo

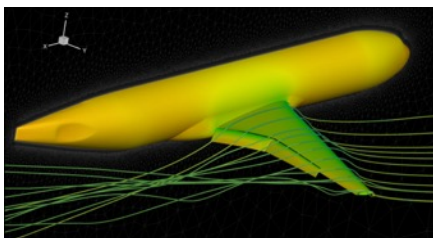
- HDR-100 InfiniBand network (100 Gb/s)
- 576 compute nodes
- 72 TiB of 8-byte values
- ~480 GiB/s (~150 seconds)

Arkouda Argsort Performance



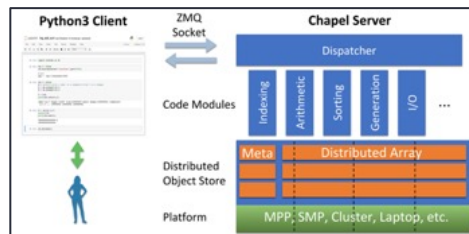
Implemented using ~100 lines of Chapel code

For more about applications of Chapel...



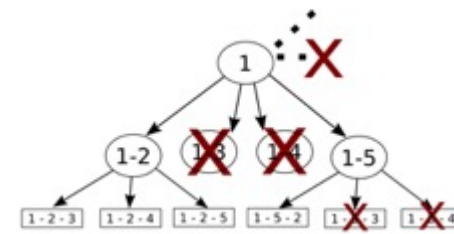
CHAMPS: 3D Unstructured CFD

Laurendeau, Bourgault-Côté, Parenteau, Plante, et al.
École Polytechnique Montréal



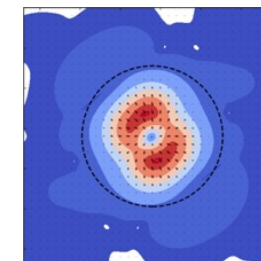
Arkouda: Interactive Data Science at Massive Scale

Mike Merrill, Bill Reus, et al.
U.S. DoD



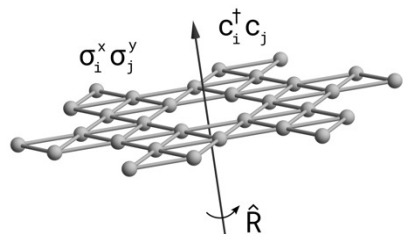
ChOp: Chapel-based Optimization

T. Carneiro, G. Helbecque, N. Melab, et al.
INRIA, IMEC, et al.



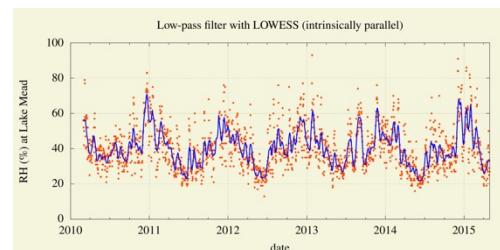
ChplUltra: Simulating Ultralight Dark Matter

Nikhil Padmanabhan, J. Luna Zagorac, et al.
Yale University et al.



Lattice-Symmetries: a Quantum Many-Body Toolbox

Tom Westerhout
Radboud University



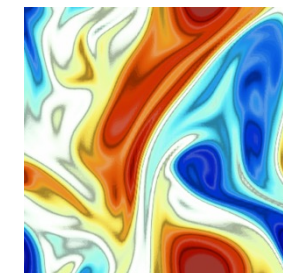
Desk dot chpl: Utilities for Environmental Eng.

Nelson Luis Dias
The Federal University of Paraná, Brazil



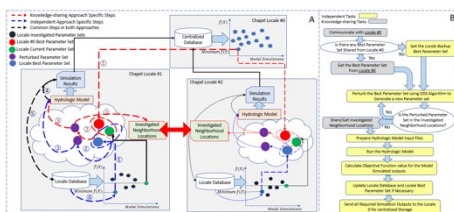
RapidQ: Mapping Coral Biodiversity

Rebecca Green, Helen Fox, Scott Bachman, et al.
The Coral Reef Alliance



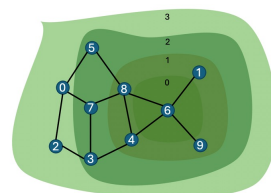
ChapQG: Layered Quasigeostrophic CFD

Ian Grooms and Scott Bachman
University of Colorado, Boulder et al.



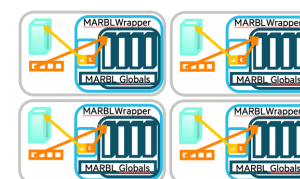
Chapel-based Hydrological Model Calibration

Marjan Asgari et al.
University of Guelph



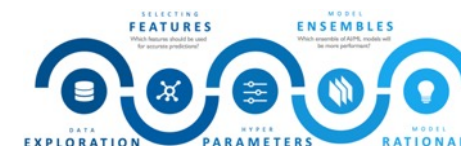
Arachne Graph Analytics

Bader, Du, Rodriguez, et al.
New Jersey Institute of Technology



Modeling Ocean Carbon Dioxide Removal

Scott Bachman Brandon Neth, et al.
[C]Worthy

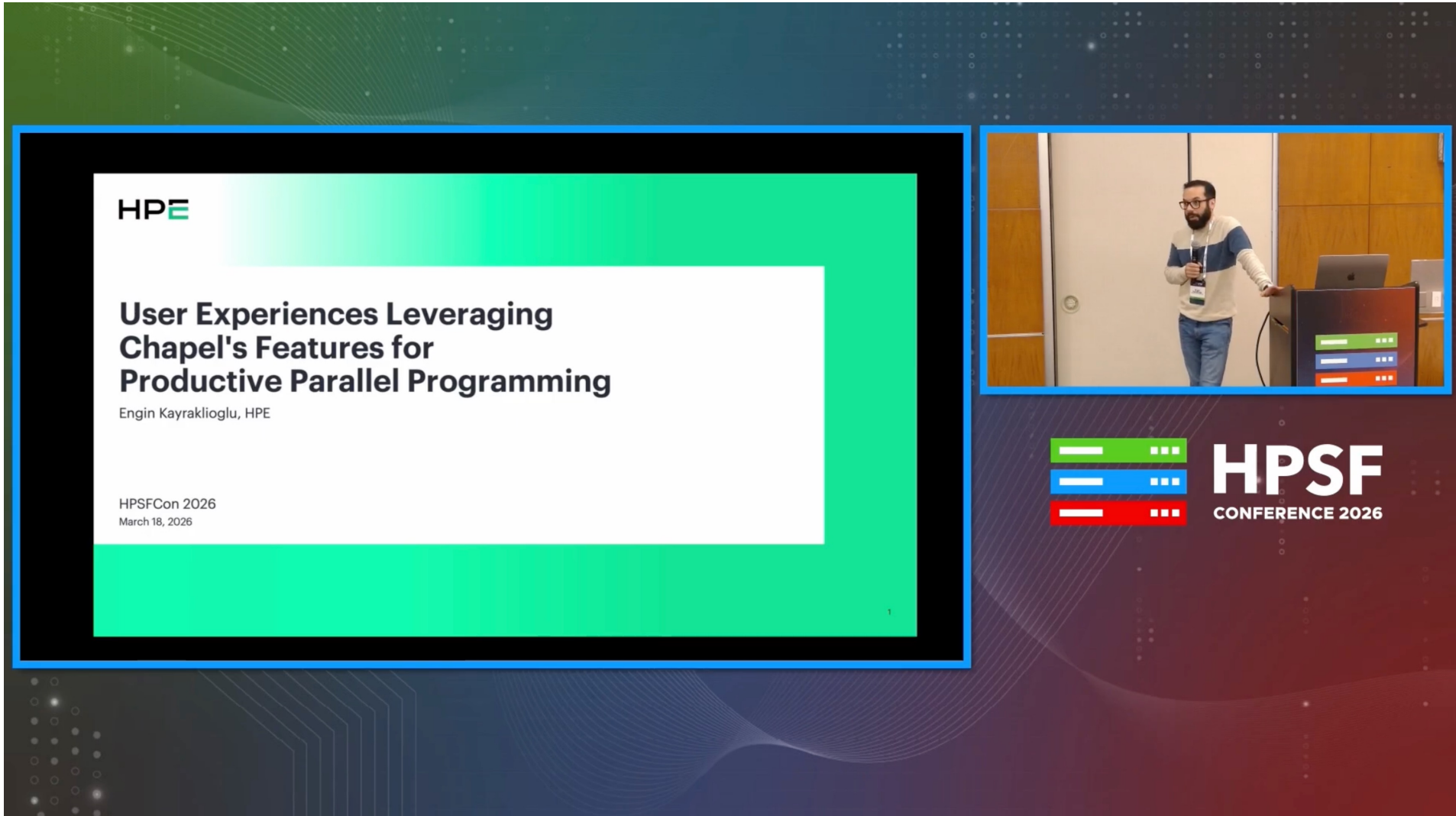


CrayAI HyperParameter Optimization (HPO)

Ben Albrecht et al.
Cray Inc. / HPE

[images provided by their respective teams and used with permission]

...see Engin Kayraklioglu's HPSFCon 2026 talk



The image shows a presentation slide for HPSFCon 2026. The slide has a green header with the HPE logo. The main title is "User Experiences Leveraging Chapel's Features for Productive Parallel Programming" by Engin Kayraklioglu, HPE. The event is HPSFCon 2026, March 18, 2026. A video inset in the top right shows Engin Kayraklioglu, a man with a beard and glasses, wearing a blue and white striped shirt, standing next to a podium with a laptop. The background of the slide features a dark blue and green abstract design with circuit-like patterns. The HPSF CONFERENCE 2026 logo is visible in the bottom right corner of the slide area.

HPE

**User Experiences Leveraging
Chapel's Features for
Productive Parallel Programming**

Engin Kayraklioglu, HPE

HPSFCon 2026
March 18, 2026

HPSF
CONFERENCE 2026

...or our “7 Questions for Chapel Users” Interview Series

Chapel’s proven to be generally applicable & attractive for:  Chapel Language Blog

- Computational Fluid Dynamics
- Earth Sciences
- Exploratory Data Analytics
- Astrophysics
- Graph Analysis
- Artificial Intelligence
- ...

About Chapel Website Featured Series Tags Authors All Posts

7 Questions for Scott Bachman: Analyzing Coral Reefs with Chapel

Posted on October 1, 2024.

Tags: Earth Sciences Image Analysis GPUs

User Experiences Interviews



7 Questions for Akihiro Hayashi: Early Chapel GPU Support through Multiresolution Abstractions

Posted on March 18, 2026.

Tags: GPUs User Experiences Interviews ChapelCon



7 Questions for Tiago Carneiro and Guillaume Helbecque: Combinatorial Optimization in Chapel

Posted on July 30, 2025.

Tags: Searching / Sorting GPUs User Experiences

Interviews



7 Questions for CHAMPS Developers: Empowering Academic R&D to Create Cutting-Edge CFD Apps in Chapel

Posted on March 26, 2026.

Tags: Computational Fluid Dynamics User Experiences

Interviews



By: Engin Kayraklioglu, Brad Chamberlain
Part of a series: [7 Questions for Chapel Users](#)

7 Questions for Éric Laurendeau: Computing Aircraft Aerodynamics in Chapel

Posted on September 17, 2024.

Tags: Computational Fluid Dynamics User Experiences Interviews



7 Questions for David Bader: Graph Analytics at Scale with Arkouda and Chapel

Posted on November 6, 2024.

Tags: Graph Analytics Arkouda User Experiences Interviews



7 Questions for Marjan Asgari: Optimizing Hydrological Models with Chapel

Posted on September 15, 2025.

Tags: Earth Sciences User Experiences Interviews



7 Questions for Nelson Luís Dias: Atmospheric Turbulence in Chapel

Posted on October 15, 2024.

Tags: Earth Sciences Computational Fluid Dynamics

User Experiences Interviews Data Analysis

By: Engin Kayraklioglu, Brad Chamberlain
Part of a series: [7 Questions for Chapel Users](#)



7 Questions for Bill Reus: Interactive Supercomputing with Chapel for Cybersecurity

Posted on February 12, 2025.

Tags: Data Analysis Arkouda User Experiences Interviews



7 Questions for Oliver Alvarado Rodriguez: Exploiting Chapel’s Distributed Arrays for Graph Analysis through Arachne

Posted on January 21, 2026.

Tags: Graph Analytics Arkouda Sparse Arrays

User Experiences Interviews

By: Engin Kayraklioglu, Brad Chamberlain
Part of a series: [7 Questions for Chapel Users](#)



<https://chapel-lang.org/blog/series/7-questions-for-chapel-users/>

Closing Thoughts



AI and Parallel Languages

Q: Now that AIs can program*, do languages like Chapel still have value?

(* = your mileage may vary)

A: I'd say "definitely", at least for the foreseeable future:

- Targeting a single, unified parallel language should be at least as successful as a mash-up of lower-level notations
- Leveraging higher-level abstractions and compiler optimizations conserves tokens by not re-solving problems
- As long as humans need to validate and maintain AI-developed code, the clearer it is the better

Chapel

```
forall (d, i) in zip(Dst, Inds) do
  d = Src[i];
```

SHMEM (Exstack version)

```
i=0;
while( exstack_proceed(ex, (i==l_num_req)) ) {
  i0 = i;
  while(i < l_num_req) {
    l_idx = pckindx[i] >> 16;
    pe = pckindx[i] & 0xffff;
    if(!exstack_push(ex, &l_idx, pe))
      break;
    i++;
  }
  exstack_exchange(ex);

  while(exstack_pop(ex, &idx, &fromth)) {
    idx = ltable[idx];
    exstack_push(ex, &idx, fromth);
  }
  lgp_barrier();
  exstack_exchange(ex);

  for(j=i0; j<i; j++) {
    fromth = pckindx[j] & 0xffff;
    exstack_pop_thread(ex, &idx, (uint64_t)fromth);
    tgt[j] = idx;
  }
  lgp_barrier();
}
```

SHMEM (Conveyors version)

```
i = 0;
while (more = convey_advance(requests, (i == l_num_req)),
      more | convey_advance(replies, !more)) {

  for (; i < l_num_req; i++) {
    pkg.idx = i;
    pkg.val = pckindx[i] >> 16;
    pe = pckindx[i] & 0xffff;
    if (!convey_push(requests, &pkg, pe))
      break;
  }

  while (convey_pull(requests, ptr, &from) == convey_OK) {
    pkg.idx = ptr->idx;
    pkg.val = ltable[ptr->val];
    if (!convey_push(replies, &pkg, from)) {
      convey_unpull(requests);
      break;
    }
  }

  while (convey_pull(replies, ptr, NULL) == convey_OK)
    tgt[ptr->idx] = ptr->val;
}
```

Summary

Parallel programming traditionally requires a mix of notations to leverage HW

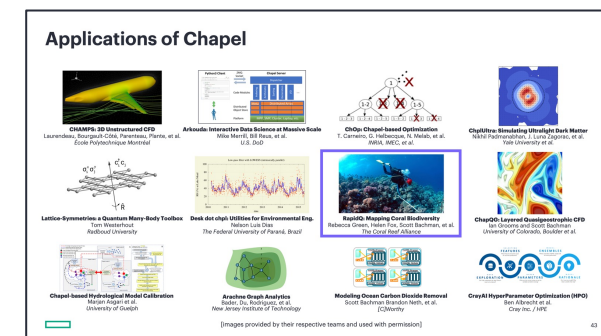
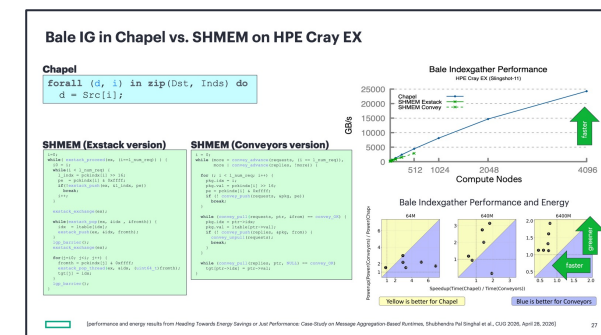
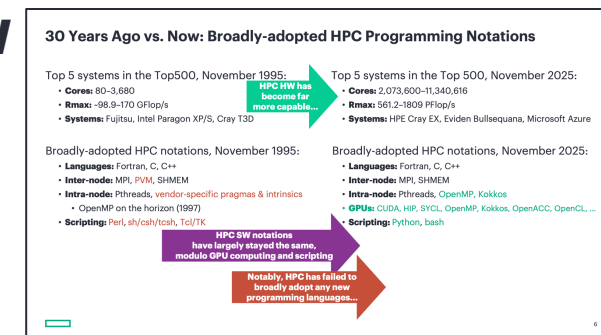
- e.g., C++ with MPI, OpenMP, and/or CUDA
- such programming models are sufficient, but leave much to be desired
- hardware has generally become more difficult to program over time
 - and at HPC scales, hardware speeds & feeds tend to dominate budgets and attention

Chapel is a unique programming language

- high-level features for parallelism and locality
- portable across CPUs, GPUs, networks, vendors, scales, ...
- scalable, performant, energy-efficient
- simplifies compiler optimizations on distributed systems (not discussed today)

Users are benefitting from Chapel, on laptops as well as supercomputers

- Exploratory Data Analysis
- Computational Fluid Dynamics
- Coral Reef image processing
- ...



Chapel Resources



Ways to engage with the Chapel Community

Synchronous Community Events

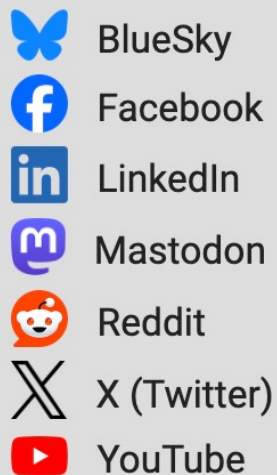
- [Project Meetings](#), weekly
- [Deep Dive / Demo Sessions](#), weekly timeslot
- [ChapelCon](#) (formerly CHI UW), annually

Asynchronous Communications

- [Chapel Blog](#)
- [Community Newsletter](#), quarterly
- [Announcement Emails](#), around big events

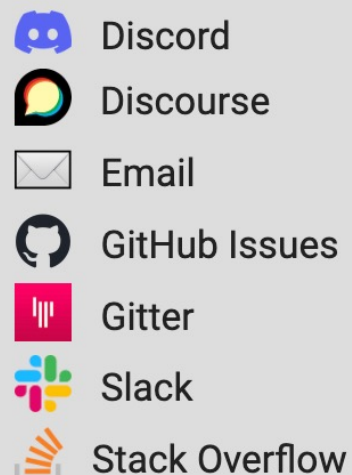
Social Media

FOLLOW US



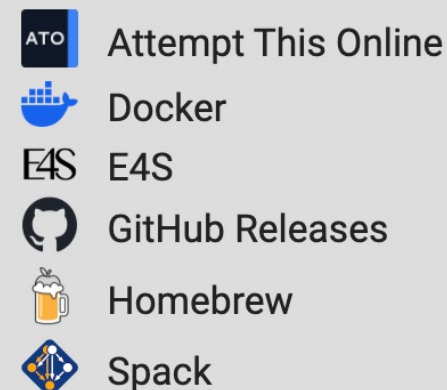
Discussion Forums

GET IN TOUCH



Ways to Use Chapel

GET STARTED

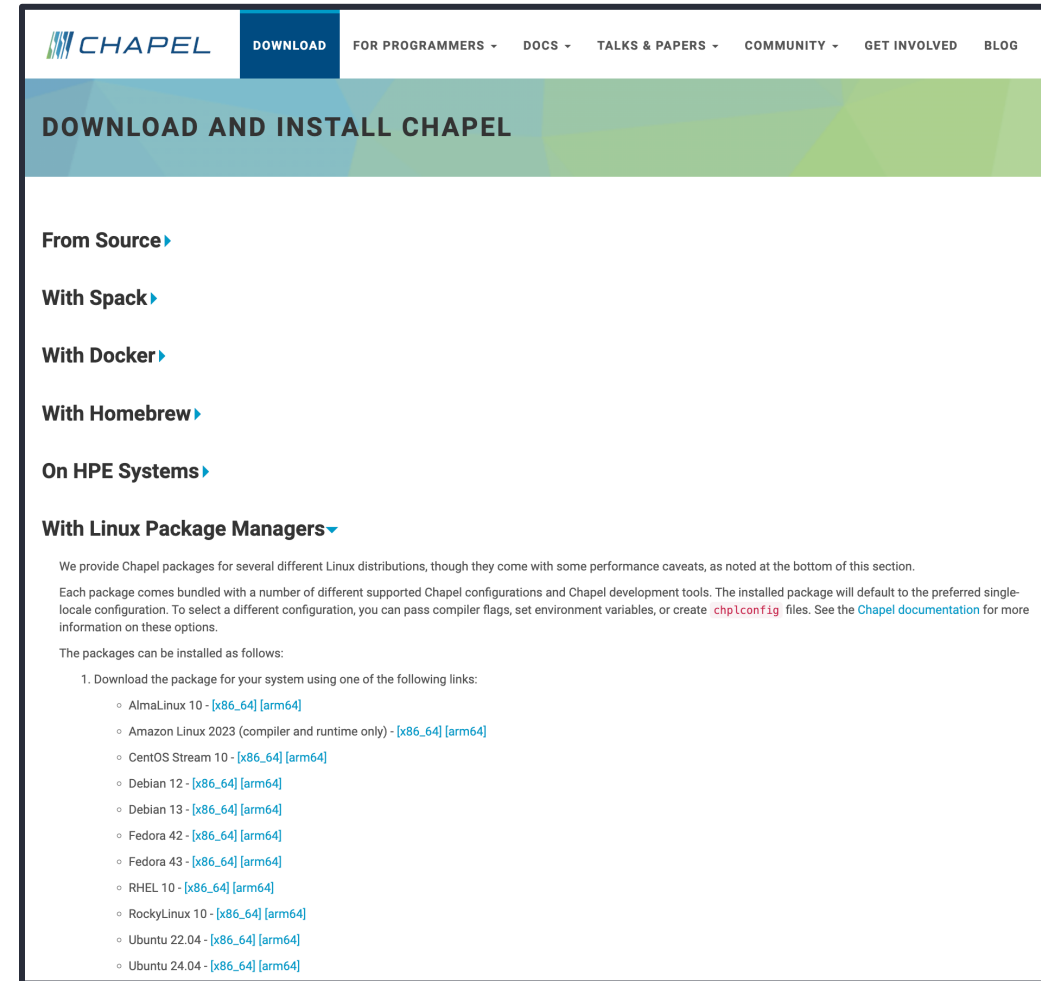


(from the footer of chapel-lang.org)

Where can I get Chapel?

Release Formats:

- Source releases via GitHub
- Spack / E4S
- Docker
- Homebrew
- Linux packages via apt/rpm
- Modules on HPE Cray systems
- Attempt This Online / GitHub Codespaces



The screenshot shows the 'DOWNLOAD AND INSTALL CHAPEL' page on the chapel-lang.org website. The page has a navigation bar with links for 'DOWNLOAD', 'FOR PROGRAMMERS', 'DOCS', 'TALKS & PAPERS', 'COMMUNITY', 'GET INVOLVED', and 'BLOG'. The main heading is 'DOWNLOAD AND INSTALL CHAPEL'. Below this, there are several sections with right-pointing chevrons: 'From Source', 'With Spack', 'With Docker', 'With Homebrew', 'On HPE Systems', and 'With Linux Package Managers'. The 'With Linux Package Managers' section is expanded, showing a list of Linux distributions and their corresponding package links. The text explains that Chapel packages are provided for several Linux distributions, each with different configurations and development tools. It also mentions that users can pass compiler flags, set environment variables, or create `chplconfig` files for more information.

DOWNLOAD AND INSTALL CHAPEL

From Source ▶

With Spack ▶

With Docker ▶

With Homebrew ▶

On HPE Systems ▶

With Linux Package Managers ▼

We provide Chapel packages for several different Linux distributions, though they come with some performance caveats, as noted at the bottom of this section.

Each package comes bundled with a number of different supported Chapel configurations and Chapel development tools. The installed package will default to the preferred single-locale configuration. To select a different configuration, you can pass compiler flags, set environment variables, or create `chplconfig` files. See the [Chapel documentation](#) for more information on these options.

The packages can be installed as follows:

1. Download the package for your system using one of the following links:
 - AlmaLinux 10 - [\[x86_64\]](#) [\[arm64\]](#)
 - Amazon Linux 2023 (compiler and runtime only) - [\[x86_64\]](#) [\[arm64\]](#)
 - CentOS Stream 10 - [\[x86_64\]](#) [\[arm64\]](#)
 - Debian 12 - [\[x86_64\]](#) [\[arm64\]](#)
 - Debian 13 - [\[x86_64\]](#) [\[arm64\]](#)
 - Fedora 42 - [\[x86_64\]](#) [\[arm64\]](#)
 - Fedora 43 - [\[x86_64\]](#) [\[arm64\]](#)
 - RHEL 10 - [\[x86_64\]](#) [\[arm64\]](#)
 - RockyLinux 10 - [\[x86_64\]](#) [\[arm64\]](#)
 - Ubuntu 22.04 - [\[x86_64\]](#) [\[arm64\]](#)
 - Ubuntu 24.04 - [\[x86_64\]](#) [\[arm64\]](#)

<https://chapel-lang.org/download/>

Where Does Chapel Run?

In the Browser:

- Attempt This Online (ATO)
- GitHub Codespaces

Laptops/Desktops:

- Linux/UNIX
- Mac OS X
- Windows (leveraging WSL)

HPC Systems:

- Commodity clusters
- HPE/Cray supercomputers, such as:
 - Frontier
 - Perlmutter
 - Piz Daint
 - Polaris
 - ...
- Other vendors' supercomputers

Cloud:

- AWS
- Microsoft Azure (?)
- Google Cloud (?)

CPUs:

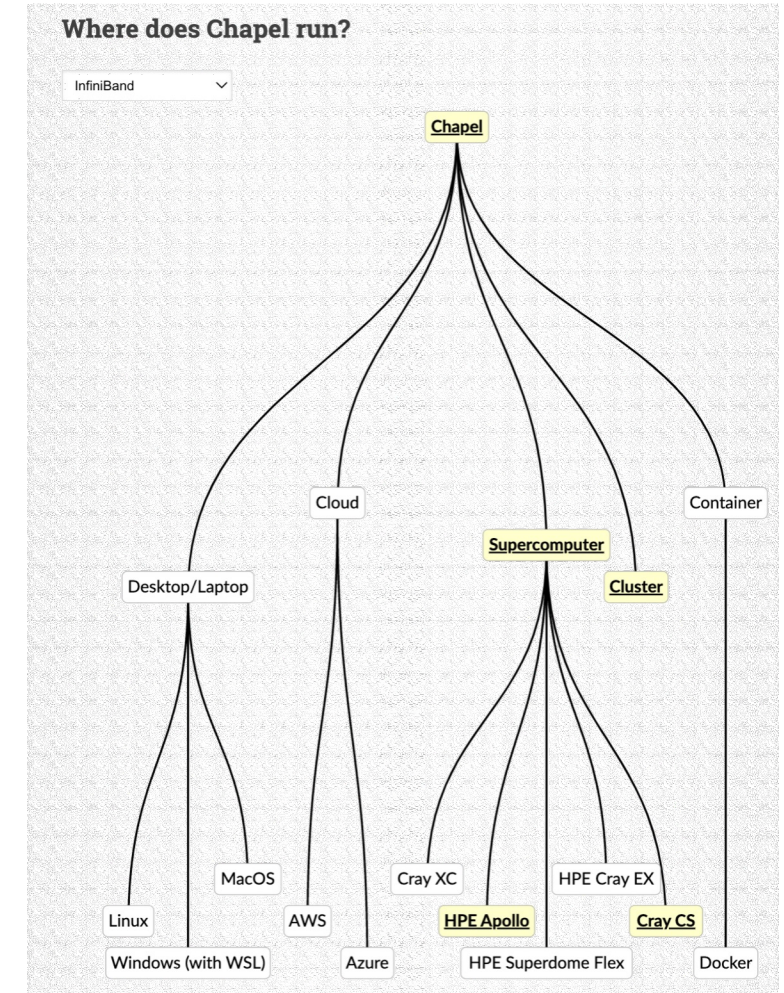
- Intel
- AMD
- Arm (M1/M2, Graviton, A64FX, Raspberry Pi, ...)

GPUs:

- NVIDIA
- AMD

Networks:

- Slingshot
- Aries/Gemini
- InfiniBand
- AWS EFA
- Ethernet



<https://chapel-lang.org/docs/usingchapel/portability.html>

Thank You

@ChapelLanguage

